Flexible, Ultra-Low Power Sensor Nodes through Configurable Finite State Machines

Juan Carlos Peña Ramos
ESAT-MICAS
K.U. Leuven
Leuven, Belgium
jcarlosp@esat.kuleuven.be

Marian Verhelst
ESAT-MICAS
K.U. Leuven
Leuven, Belgium
marian.verhelst@esat.kuleuven.be

***Abstract*— Due to the recent popularity of context-sensitive applications, there is a growing need for reliable, long-lifetime ubiquitous sensor nodes. The severe energy-efficiency requirements of these energy-scarce devices require complementing traditional circuit-level energy saving techniques, with architecture-level methods. Traditional approaches such as exploiting parallelism have however limited impact in sensor node processors, due to their control-dominated and event-based, irregular data processing workload patterns. Executing event-based tasks in specialized finite state machines relieves the on-board microcontroller, however, at the penalty of reduced post-manufacturing configurability. An architecture proposal for configurable finite state machines assisting sensor node processors is presented, which allows saving energy through task off-load while maintaining system flexibility. Simulations demonstrate 46% energy savings when compared to a sensor node that executes tasks in a microcontroller. This gain comes at relatively minor area overhead.**

Keywords—Configurable finite state machines, configurable datapath, event based processing, ultra-low power sensor.

I. INTRODUCTION

It is through sensors that an application can observe the physical world, process data, and make decisions based on these measurements. In some cases only one accurate sensor is needed while in others, data is gathered through a network of smaller low-cost sensors, which are often connected wirelessly. Regardless of their degree of specialization, and whether they can be found in cellphones, medical equipment, cars or houses, sensors are starting to permeate every aspect of a user's environment, where they need to sense and process as much data as possible before their batteries are depleted and need to be recharged. In cases where conventional charging is not possible and energy is harvested from the environment, the energy budget of the application might be even more stringent. As a result, keeping energy consumption to a minimum is a critical design parameter when designing sensor nodes.

Focusing on the energy consumption of the processing subsystem of the sensor node, several energy saving techniques have been proposed. These are mainly traditional circuit-level techniques which have been ported to sensor node applications, such as advanced power gating, clock gating, or voltage and frequency scaling techniques (see Section II.B). Also several architectural modifications have been proposed, e.g. introducing dedicated accelerators and increased parallelism to off-load the main processor core (see Section II.C). However, the impact of these approaches is limited in common sensor node applications characterized by control-dominated and event-based, irregular workload patterns.

Recently, it has been shown that significant energy can be saved in typical sensor node scenarios, by selectively off-loading control tasks to dedicated finite state machines outside the core [6]. This however comes at the inacceptable cost of losing all post-manufacturing configuration capabilities. This work will advance the state-of-the-art concerning this concept through the introduction of configurable FSMs, in order to exploit the introduced energy savings opportunities, while maintaining sufficient flexibility. To this end, this paper will both introduce a parameterized architectural framework for configurable FSMs for sensor node task off-load, as well as assess the energy and area penalties of the introduced flexibility to find the best trade-off in terms of system configurability.

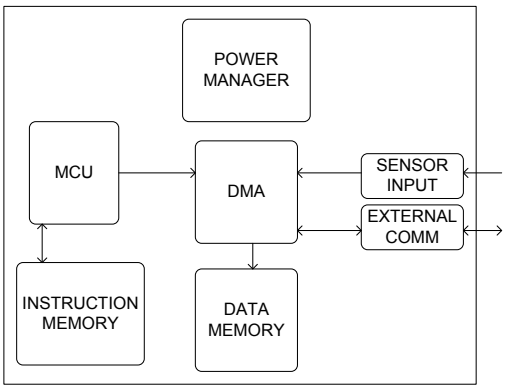
This paper is organized as follows: Section II briefly introduces the main functional blocks of the targeted sensor node system, and reviews state-of-the-art energy saving techniques in sensor node processors at circuit and architectural level. Subsequently, Section III presents the concept of Configurable Finite State Machines, their operating principle and architecture. A cost analysis in terms of area is done based on the configuration parameters of the architecture. A case study based on an ECG sensor node is used in Section IV to quantify the obtained results in a realistic setting. Finally, Section V concludes the paper and discusses future application opportunities.

II. SENSOR NODE OVERVIEW

Before introducing the configurable FSMs and assessing their benefits, it is important to understand the architecture and energy consumption of a sensor node.

978-1-4673-6180-4/13/$31.00 ©2013 IEEE

A. Architecture of a sensor node



Fig, 1 Main functional units of a sensor node

In order to understand how the architectural changes proposed in this document affect a sensor node it is important to briefly introduce its main functional units, which are shown in Fig, 1. Sensor nodes usually are centered around a low-power microcontroller unit (MCU) with local instruction and data memories, which is in charge of the data processing and general management. This is complemented by an interfacing subsystem with the sensing element (electrodes, accelerometers, etc.), a subsystem to communicate externally (wireless or wired) and a power management unit, e.g. executing power gating and frequency scaling strategies. A direct memory access (DMA) module is often introduced to share the data memory between the MCU and the input and output subsystems and allow efficient data movement without MCU involvement.

The input sensor typically needs to be activated frequently in order to measure external parameters. The same applies to the external communication module, if it is assumed to be reactive to commands received from an external agent. The amount of time the microcontroller needs to be active hence depends not only on the time it takes to process the sensed data, but also on how involved it is in the sensor input and communication tasks. It is important to note that even in systems that use low power microcontrollers, SRAM memories consume a major portion of the total system energy [2]. This is particularly important in sensor nodes where most of the area and thus leakage energy that is dissipated come from the SRAM blocks in the system.

B. Circuit level energy savings in sensor node processors

Energy consumption in sensor nodes can be separated in that consumed by the analog components (sensors, transmitters) and that consumed by the digital circuitry (signal processing, communication protocols), this work focuses on the latter. Energy consumed by digital circuits can again be split into dynamic and static energy. Dynamic consumption is energy spent in the switching of transistors and can be described as $\boldsymbol{CV_{DD}^2}$ where $\boldsymbol{C}$ is the equivalent switching capacitance of the digital circuit, $\boldsymbol{V_{DD}}$ is the supply voltage and $\boldsymbol{f}$ is the operating frequency. Static power consumption is caused by leakage currents in a CMOS gate, depends exponentially on the difference between the gate-source voltage V_{GS} and threshold voltage V_T, and is independent of the frequency and switching activity of a gate. The problem of static energy grows with each new generation of CMOS process technology, as V_T is scaled down.

Traditional circuit-level energy saving techniques for energy-efficient processors, can be split in design-time and run-time techniques. Common design time techniques to reduce dynamic power consumption are the use of multi-Vdd, and multi Vt technologies. Multi-V_{DD} approaches consist of lowering the V_{DD} on certain non-timing-critical blocks. This, however, does not only make the delay in the block with reduced voltage slower, but also increases complexity in the communication between blocks with different voltages. On the other hand, multi-threshold (Vt) logic, uses transistors with high V_T (and thus lower leakage) for sections that do not require high performance, and transistors with low V_T for modules that do. Most libraries offer two or three V_T versions of their cells.

More aggressive saving techniques adapt circuit parameters at run-time to achieve increased savings. The most drastic approach, targeting the reduction of leakage power, is power gating. The principle is simple, shutting down the power supply of an unused logic block. Its implications are however significant, as it affects communication between blocks, the amount of time needed to enter and exit power gated modes, and the complexity that this adds to the overall control of the module. This is often complemented with aggressive methods of voltage scaling. Supplying an operating voltage close to the threshold level provides significant energy and leakage power reduction in logic and SRAM circuits, but suffers from strong process variations requiring advanced variation mitigation strategies [2]. Several of these techniques have been applied successfully in wireless sensor network (WSN) applications, such as dynamic voltage scaling (DVS) [3] and dynamic voltage and frequency scaling (DVFS) [4], in which the supply voltage and/or processing frequency is varied depending on the current circumstances.

A much more detailed explanation of methods to reduce static and dynamic power consumption at circuit level can be found in [1].

C. Architectural energy savings in sensor node processors

It is well known that through hardware customization and specialization at the architectural level, energy efficiency can be greatly improved. These techniques mainly focus on applications that are data-flow oriented with relatively simple control flows, and try to exploit the parallelism in compute-intensive kernels by matching it at hardware level. This reduces execution time, which allows to lower the operation frequency of the design. By having more relaxed timing constraints, the design can operate at a lower supply voltage, and hence save dynamic energy.

Exploiting parallelism might, however, not be as useful in the context of sensor nodes, where nodes usually remain inactive for long periods of time. As a node is typically triggered by external events, its processing is of irregular nature and control-flow rather than data-flow dominated [5]. Even though the processing of the sensed values can be improved through the traditional techniques mentioned before,

the resulting reduction in energy consumption will be less significant, as the contribution of static leakage currents is now much more dominant when compared to dynamic energy consumption.

In sensor nodes, the processor's energy is mainly dissipated in its local memories [1]. An interesting approach presented in literature [6] is to off-load event-related tasks from the microcontroller to hardwired finite state machines (FSMs), which can execute control tasks in a much more efficient manner. By doing this, the microcontroller can be power gated most of the time, and the memories kept in data retention mode to minimize static power consumption. This approach brings opportunities towards a reduction of energy consumption with two orders of magnitude [5], but has the disadvantage that FSMs are hardwired into the silicon. The latter inhibits any post-production changes to the control flow, making it impossible to correct bugs, comply with standard updates, or change the node's behavior for any other reason.

The contribution of this work is to go one step further by making these FSMs configurable, in order to maintain the advantages mentioned above while keeping sufficient flexibility. By making FSMs configurable there is, however, an area and thus leakage overhead compared to the hardwired implementation. The goal of this work is two-fold: 1.) Introduce a parameterized architectural framework for configurable FSMs for sensor node task off-load; 2.) assess the energy and area penalties of the introduced flexibility to find the right degree of how configurability for the introduced FSMs.

III. Configurable Finite State Machine (CFSM)

A. Operating principle

Traditional, fixed FSMs have the following components:

- A set of states: Only one can be active at a particular point in time, and a state will have one or multiple connections to other states.
- A set of inputs: Depending on the current state, a transition to the next state is triggered by certain inputs.
- Transition logic: Each state has a set of transitions, either to itself or to other states. The transition logic determines the next state based on the current state and the inputs to the FSM.
- Set of outputs: The outputs of the FSM. These outputs can either depend on the inputs and the current state (called a "Mealy FSM"), or only depend on the current state (hence called a "Moore FSM"). The implementation presented here focuses on Moore FSMs.

The first step in building a Configurable FSM (CFSM) is to determine which parameters should be flexible after manufacturing. First of all, a flexible FSM requires the transition logic to be programmable, so that the user can modify the transitions between states. Secondly, the output logic must also be programmable, to determine the relationship between the state and the FSM's output. Finally, it has also been noted that many state machines rely on counters to e.g. wait for a certain amount of cycles or to execute a particular task. Hence, a set of programmable counters has to be included.

All this configuration information has to be stored in a local memory or register file. The maximum number of states will determine the amount of memory needed, and should be kept as small as possible, while maintaining flexibility. The architecture introduced in Section III.B, therefore contains several measures to reduce memory footprint, without affecting programmability where required. This involves the introduction of a default transition from every allowed state, which keeps the transition logic simple, as well as limiting the maximum number of transitions that each state can have.

B. Architecture of a CSFM

To fulfill the configuration requirements stated above, the proposed implementation consists of the hardware blocks shown in Fig. 2. The modules in the architecture can be broadly divided in two groups, the ones that provide configurability to the FSM, and the logic that is present in any FSM.

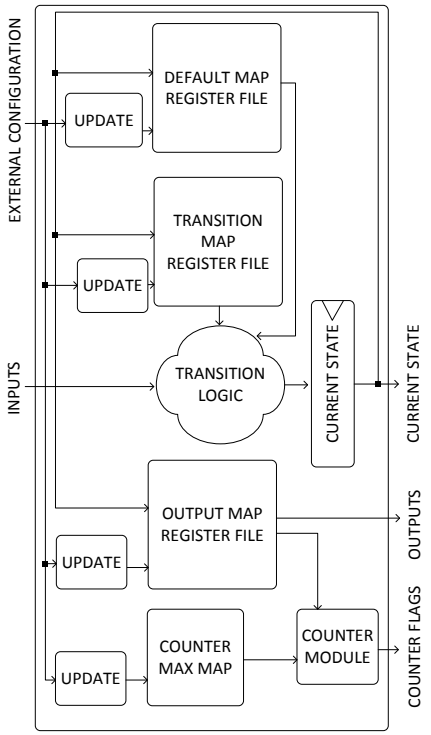


Fig. 2 CFSM main operating blocks

The FSM is configured through three register files, from now on referred to as "maps", so that the states, transitions, counter configuration and outputs can be modified post-production by the user: These maps (except the counter configuration) have one entry per state, and are addressed by the current state.

- Default map (DMAP): A register file that stores the default transition for each state. By having one default transition, the transition logic and the transition map can be greatly simplified.
- Max counter map (CMAX): In this register file different maximum values for the counters are stored. This map is not addressed by the current state as it has one entry per counter in the architecture, and is always connected to it.
- Transition map (TMAP): A register file that stores the configuration of the transition logic for each state. How this is implemented will be explained in detail in Section III.C.
- Output map (OMAP): A register file that stores the output of each state. These outputs directly communicate with the external hardware modules. It also has bits that configure the behavior of the counters in each state (increment, reset).
- Update modules: Through this interfaces the maps in the CFSM can be reconfigured by the microcontroller or by a boot loader module directly from memory.

The rest of the modules in the design are present in most FSM implementations (current state register, transition logic, counters) and are not configurable by the user post-production.

- Transition module: A combinational module that depending on the current output of the Transition map and the inputs determines the next state.
- Current state register: A register holding the current state value. This value is moreover used as the address for all the maps to extract currently relevant entries. It is also an output of the CFSM, in case external logic can benefit from knowledge of the current state.
- Counter module: One or more counters can be added to the design (not configurable in post-production); they read the Max counter map and raise a status flag when the counter overflows. This flags can be used with the inputs in the transition module to produce the next state.

The viability of replacing a hardwired FSM with a CFSM depends then on its size and energy consumption penalties, which will be mainly dominated by the size of DMAP, TMAP and OMAP. The parameters that we will use to describe the architecture and quantify these penalties are the following: STATE_N represents the maximum number of states in the CFSM, TRIGGER_N represents the number of input bits, OUTPUT_N represents the number of output bits, COUNTER_N represents the number of counters and thus the number of output counter flags that will be available in the CFSM, COUNTER_W represents the maximum value a counter can reach, and TRANS_N represents the maximum number of transactions per state.

The look-up tables of DMAP, OMAP and TMAP have STATE_N entries. The width of each entry is defined as follows:

- As DMAP stores only the index to the default transition state for each state, it requires $\log_2$(STATE_N) bits.
- OMAP stores for every possible state, all the outputs of the CFSM, so each entry has OUTPUT_N bits.
- In order to define the width of the TMAP, on more detailed explanation is needed on the operating principle of these transitions and their configurability.

C. Transitions and triggers – Compromises in flexibility

In a regular FSM a transition from one state to another can occur depending on a certain combination of its inputs (or internal counters). Some transitions can be triggered by one input, while another might be triggered by a combination of multiple bit words. In order to keep all options open in the transition definition, the transition logic would have to be utterly flexible, similar to an FPGA-like structure. The amount of bits to configure this, and resulting area and energy penalty, would be unacceptably large.

A compromise must hence be made between the configurability of the transitions, and the resulting complexity overhead. In this regards, two options will be explored:

1.) State transitions can only be triggered by single bit inputs values, but the amount of possible transitions per state is not limited (only limited by the number of input bits, TRIGGER_N).

2.) Multi-bit combinations of input are supported as state transition triggers, but limiting the amount of possible transitions per state.

In the first option, every state transition of the CFSM is triggered by the value of one bit of the input triggers. The transition map hence stores for every current state, and for every one of the TRIGGER_N input bits, which will be the next state in case it is asserted.

An implementation as shown in Fig. 3, uses this transition table data to compute the next state: The input bits are demultiplexed to the OR-gate of the next state they are triggering for. Each demultiplexer of active triggers therefore generates a one-hot word representing the next state. The amount of bits needed to configure each demultiplexer is hence $\log_2$(STATE_N+1), with the extra bit required to signal if the trigger is not used in the current state. The bitwise *or* of these words gives the one-hot representation of the next state. If the configuration is not done properly and the one-hot word has two active bits, an error flag should be activated. If every no trigger produces a transition and the one-hot word is zero, then the default transition from DMAP is used. The number of bits that each state hence requires in TMAP is $\log_2$(STATE_N+1) ·TRIGGER_N.

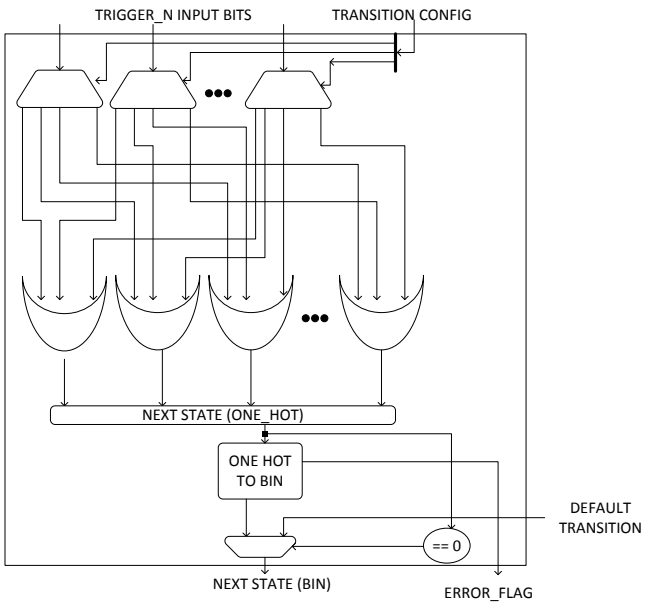


Fig. 3 Transition module architecture

A second option is to allow multi-bit combinations to trigger state transitions. This requires storing a mask word of length TRIGGER_N for every allowed state transition. As it would be unfeasible to do this for every other state in the FSM, the maximum number of possible next states is limited in this implementation to TRANS_N. As it should remain programmable which next states these are, the state that each of the TRANS_N transitions leads to would have to be stored as well. Hence, a transition configuration map with a length of $\log_2(TRANS\_N + 1) \cdot TRIGGER\_N$ would be needed to store the trigger masks, and a word with length of $(TRANS\_N) \cdot \log_2(STATE\_N)$ to store the next state for every trigger combination. Unless the number of transitions is very small, this ultimately leads to a larger number of registers in the end. Two examples of this can be seen in Fig. 4, as two implementations (FSM0, with 32 states, 8 inputs and 4 outputs and FSM1 with 10 states, 4 inputs and 4 outputs) are compared between options 1 and 2. It is clear that for option 2 the area consumption grows quickly with the maximum number of transitions. As a result, option 1 is selected and implemented in this work.

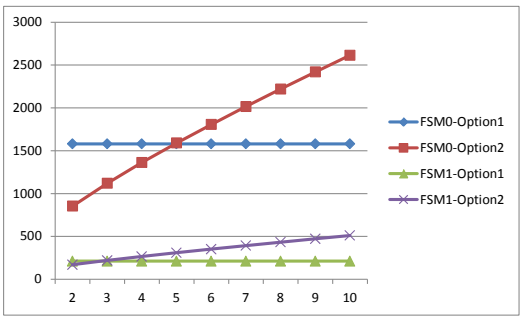


Fig. 4 Architecture comparison in number of flip-flops

The selection of this single-bit implementation option means that any necessary multi-bit triggers have to be consolidated outside the configurable FSM into single bit triggers that the CFSM can handle. This can e.g. be done in a hardwired way. Although this significantly reduces the CFSM cost, one can argue that this limits the power of the CFSM to be able to adapt to post-production updates. However, the behavior of the CFSM can still be modified and new states can be added, as long as the same triggers are used with regard to the initial implementation, which still covers a wide range of post-manufacturing updates, like changes in a startup sequence or in the implementation of a serial protocol. The designer can decide himself to provide additional triggers to keep the option of using them in the future open, or can even decide to implement a small configurable trigger generation block outside the CFSM in case that flexibility is desired. A similar trade-off can be made regarding the number of output bits OUTPUT_N supported by the CFSM. The designer can decide to have a reduced number of configurable outputs, and depend instead on the current state output using the output bits for future expandability (see e.g. ECG use case).

D. Cost evaluation

Table 1 summarizes the total required number of registers required by the architecture:

TABLE 1 REGISTER COST OF DESIGN

Module name	Total number of flip-flops
DMAP	$STATE\_N \cdot \log_2(STATE\_N)$
TMAP	$STATE\_N \cdot \log_2(STATE\_N+1) \cdot TRIGGER\_N$
OMAP	$STATE\_N \cdot (OUTPUT\_N+(COUNTER\_N*2))$
CMAX	$COUNTER\_N \cdot COUNTER\_W$
Current state register	$\log_2(STATE\_N)$
Counter register	$COUNTER\_N \cdot COUNTER\_W$

Table 2 shows the cost in terms of register count increase of introducing an additional state, additional input, or additional output bit into the CFSM.

TABLE 2 REGISTER COST OF INCREASING A PARAMETER

Increase in Parameter	Cost in flip-flops
State	$\log2(STATE\_N) \cdot (1+TRIGGER\_N)+OUTPUT\_N$
Input	$STATE\_N \cdot \log2(STATE\_N+1)$
Output	STATE_N

IV. CASE STUDY: ECG SENSOR NODE

A. F0: Basic sensing scenario

In order to assess the effects of unloading tasks from the MCU to FSMs, and assess the energy and area cost of the introduction of CFSMs, an sensor node running an electro-cardiogram (ECG) analysis task was simulated. The node receives data from a sensor through an I2C module at a fixed rate. The MCU interpolates the signal, executes the Pan-Tompkins algorithm for QRS peak detection [7], first executing some filtering and then searching for peaks in the processed signal to detect the heart rate, while detecting and discarding "noise" peaks. The node needs to communicate the average number of samples between peaks once every minute. Its architecture is depicted in Fig. 5. The operating scenario where all tasks described above are executed in the core, will be denoted as scenario 'F0'.

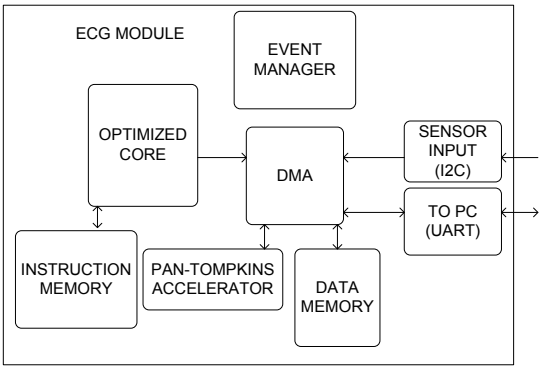


Fig. 5 ECG module architecture where FSMs handle event related tasks.

B. F1: Offloading to fixed FSMs

To assess the effectiveness of off-loading control-oriented tasks to fixed FSMs, the following tasks were off-loaded from the MCU to dedicated hardwired units:

- The I2C module, for the ECG sensor input
- The UART master module, to communicate with the PC
- The event manager, which controls which modules are clock gated and also acts as control for the DMA, deciding who communicates with data memory, and who communicates with the MCU.

This off-loading allows keeping the micro-controller more frequently in sleep mode (clock gated, but not power gated in our implementation), with its memories in data-retention mode (lower leakage dispersion but data is not lost). While the processor is asleep, the event manager is responsible to configure the DMA to allow the sensor input module access to data memory. Every time a new sample is detected, the event manager wakes up data memory, writes the sample, and powers it down. When a predefined number of samples have been captured, the event manager powers up instruction memory, configures the DMA to give the core access to data memory, and wakes up the core. Once the samples have been processed, the core sends the average number of samples between peaks to the PC to reveal the computed heart rate. The core subsequently enters sleep mode, and the event manager clock-gates it, and puts both instruction and data memories back into data retention mode. The Event manager then goes again to wait for more samples. This scenario with fixed FSM off-loading will be denoted by scenario ‘F1’.

C. F2: Pan-Tompkins accelerator with fixed FSM

The bulk of the remaining processor tasks relate to processing the ECG signals to extract the heart rate using the Pan-Tompkins algorithm. External data accelerators are often suggested to execute such processing more efficiently. To this end, a configurable data path was implemented, which could be programmed to serially perform several subtasks of the Pan-Tompkins algorithm, consisting of a low-pass filtering stage, a high-pass filtering stage, derivative computation, squaring and finally an integration. The control of this accelerator, to repeatedly set all its configuration bits to configure it to the correct sequence of operating modes to implement Pan Tompkins, is implemented first as a hardwired FSM. In this usage scenario, the core hence has a configurable datapath, with a fixed FSM to control it, shown in Fig. 6. The core goes to sleep every time the Pan-Tompkins algorithm is processing, and wakes up when done, to execute the remaining peak search. This scenario is denoted as ‘F2’.

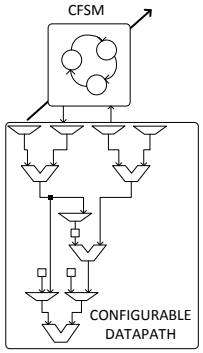


Fig. 6 Pan-Tompkins configurable datapath and control

D. F3: CFSM in Pan-Tompkins accelerator

The final step is to replace one of the hardwired FSMs with a CFSM. In this section, replacing the FSM in the Pan-Tompkins accelerator will be discussed, as it leads to another example where the configurability of the CFSM needs to be balanced carefully against its register count.

The hardwired FSM has nine states: IDLE, PUSH, LOWPASS, HIGHPASS, DERIVATIVE, SQUARING, INTEGRAL, REGISTER SHIFT, AND FINISH, there is only one trigger, which signals a new value to process. The particular thing about this FSM is that it requires configuring the datapath of the accelerator, which is done through 22-bit word. This would make OMAP quite big, while TMAP and DMAP are very small in comparison.

As discussed in Section III.C, the designer can decide to work with a smaller set of output bits, and construct part of the configuration signal come externally to the CFSM based on the current state register. The presented implementation did this for the configuration bits of the accelerator, leaving four more in the output register for future expandability. The resulting FSM is very small, although the pattern of the operations can still be changed. E.g., if the algorithm were to change to remove or repeat some of the algorithmic stages, so that the CFSM could be changed to cycle through the states in a different order. This trade-off can be made by the designer at design time. The resulting operating scenario, using the configurable FSM is denoted as scenario ‘F3’.

E. Results in terms of area, performance and energy

The four implementation scenarios were simulated using TSMC 40 nm technology at 1.1 volts, operating at 10 Mhz. They will be compared quantitatively in terms of their area and energy implications on the sensor processing subsystems. This will allow to assess the energy saving opportunities of event off-loading to external FSMs, as well as to assess the overhead of introducing configurability into these FSMs.

- F0: No task off-loading, everything implemented on the micro-controller core.
- F1: Event related tasks off-loaded from core to fixed FSMs.
- F2: Scenario F1, with additional data processing off-loaing to accelerator with fixed FSM.
- F3: FSM in accelerator is replaced by CFSM.

For each of these scenarios, Table 3 lists the total number of combinational area and registers in the design (without taking into account memories), The total area estimated, including the area of data and instruction memories. Two energy measures are shown, the first is for the processing of a batch of samples (in this case 66), which depending on the implementation might take more or less time, and the second is the total energy consumption in a second, taking into account the time the module is idle waiting for new samples.

TABLE 3 RESULTS SUMMARY

	F0	**F1**	**F2**	**F3**
Combinational area (gates)	5847	6203	8199	9013
Non-combinational area (gates)	3658	3975	15142	17438
Total area incl. memories(mm^2)	0.2460	0.2468	0.2626	0.2663
Energy consumed processing 66 samples (uJ)	8.8	8.79	1.50	1.51
Power consumption (uW)	30.106	22.95	16.018	16.022

As can be seen from the previous table, unloading only control tasks from the processor with fixed FSMs (Comparing F0 and F1) reduced energy consumption by 23% (Energy spent in one minute), even though the energy consumed by the data processing remained the same. By also unloading the bulk of the data processing to the accelerator (comparing F2 and F1), there was a significant reduction in the energy consumed to process the sensor data (x5.8, energy per batch of 66 samples), however, due to the large amount of leakage dissipated by the memories, the overall improvement in energy consumption was much smaller (x1.37 more efficient in overall energy per second). Finally, the table shows that by replacing the hardwired FSM for the CFSM does not significantly impacts area or performance, so the benefits of the hardwired FSM are kept while increasing flexibility. Comparing the energy spent in one second in F0 and F3, a reduction of 46% can be seen.

V. CONCLUSIONS AND FUTURE WORK

In this work introduced an architecture for configurable FSMs targeting the off-load of control tasks from sensor node processors in order to save energy, without giving up all programming flexibility. The paper assessed the tradeoffs of unloading tasks to hardwired FSMs, and the overhead of making them configurable. By off-loading event-related tasks from the core, it can be kept in low power mode as much as possible. In our case study, a reduction of 46% of the consumed energy in a given period of time was observed by using techniques at the software and architectural level, without implementing power gating and other gate level energy saving methods, which can still be combined with the proposed methodology.

Our future work will assess the gains of combining the presented techniques with power gating and DVFS. More case studies are planned to assess energy saving opportunities in promising applications:

a. *Configurable I/O module:* I/O ports in an ASIC can sometimes be a scarce resource. Often, the same set of ports is multiplexed between different communication and debug modules (e.g. I2C, JTAG, USB, etc.). A CFSM might come in handy to support such multiplexing between different protocols in an area efficient way, and allow to make changes or updates to them after manufacturing.

b. *Configurable event/power manager:* By making the power/event manager module configurable, the startup sequence and main control of a SoC can be flexible and much easier to debug. Currently, such flexibility can only be achieved by implementing node event- and power-control in a core with full instruction memory, representing significant energy overhead.

ACKNOWLEDGMENT

The authors want to acknowledge Synopsys for their support with the tool Processor Designer.

VI. BIBLIOGRAPHY

[1] M. Keating, D. Flynn, R. Aitken, A. Gibbons and K. Shi, Low Power Methodology Manual For Systen-on-Chip Design, Springer, 2007.

[2] J. Kwong , Y. K. Ramadass, N. Verma and A. P. Chandrakasan, "A 65 nm Sub-Vt Microcontroller with Integrated SRAM and Switched Capacitor DC-DC Converter," *IEE Journal of Solid-State Circuits,* vol. 44, no. 1, 2009.

[3] W. Tuming, Y. Sijia and W. Hailong, "A Dynamic Voltage Scaling Algorithm for Wireless Sensor Networks," in *International Conference on Advanced Computer Theory and Engineering*, 2010.

[4] V.-T. Hoang, N. Julien and P. Berruet, "Design under Constraints of Availability and Energy for Sensor Node in Wireless Sensor Network," in *Conference on Design and Architectures for Signal and Image Processing*, 2012.

[5] M. A. Pasha, S. Derrien and O. Sentieys, "Ultra low-power FSM for control oriented applications," *International Symposium on Circuits and Systems,* pp. 1577-1580, May 2009.

[6] M. A. Pasha, S. Derrien and O. Sentieys, "System-level synthesis for wireless sensor node controllers: A complete design flow," *ACM Transactions on Design Automation of Electronic Systems,* January 2012.

[7] J. Pan and W. J. Tompkins, "A Real-Time QRS Detection Algorithm," in *IEEE Transactions on Biomedical Engineering*, 1985.